DANISH ATOMIC ENERGY COMMISSION,
Research Establishment Risø,
Reactor Physics Section,
Computer Group.

November 1962.
SA-3/1.
100 copies.

Calculation of the Value of a Determinant.

ALGOL - procedure: DETERMINANT.

1. Scope.

The following procedure calculates the value of a determinant of the order n $\geq$2. The determinant is not altered by the procedure. A shortening of the running time is obtained if many numbers in the determinant are zero.

2. Method.

The calculation method is based on the elimination of the elements below the diagonal, so the value of the determinant is simply the product of the elements in the diagonal.

If an element in the diagonal is zero, the row will be interchanged with another row and the sign of the determinant will be changed.

The elimination process proceeds from left to right. If, during the elimination process a coulomn where the elements below and on the diagonal equals zero is obtained, the value of the determinant is zero.

3. Use of the Procedure.

The procedure will be copied into the program in the place where there is written:

comment library DETERMINANT;

The order must be n $>$ 2.
The procedure call can be of the well known type:
q := DETERMINANT (D,8);
q will then obtain the value of the determinant D of the 8 th. order.

4. Algoritm.

```
real procedure DETERMINANT (A,n);
value n; integer n;
real array A;

comment : A.E.K. - November 5 th 1962 - this procedure evaluates a
determinant defined by the matrix A[1:n, 1:n] as described in SA - 3;

begin
real k,D;
integer p, q, m;
real array a[1:n, 1:n];
for p := 1 step 1 until n do
for q := 1 step 1 until n do
a[p,q] := A[p,q];
D := 1;

for p := 1 step 1 until n do
begin
m := p;
if a[p,p] ≠ 0 then go to elimination;
for m := m + 1 step 1 until n do
if a[m,p] ≠ 0 then go to interchange of rows;
DETERMINANT := 0; go to end of procedure;

interchange of rows:
D := - D;
for q := p step 1 until n do
begin k := a[m,q]; a[m,q] := a[p,q]; a[p,q] := k; end;

elimination:
for m := m + 1 step 1 until n do
begin
k := a[m,p]/a[p,p];
if k = 0 then go to next row;
for q := p + 1 step 1 until n do
a[m,q] := a[m,q] - k × a[p,q];

next row:
end of m;

D := D × a[p,p];
end of p;

DETERMINANT := D;
end of procedure:
end;
```

Ivar Balslev.
Helge Vilstrup.

DANISH ATOMIC ENERGY COMMISSION,
Research Establishment Risø,
Reactor Physics Department,
Computer Group.

Feb. 1967
SA-10/1
50 copies.

Solution of systems of linear equations.

ALGOL - procedure: CROUT.

1.Scope.

The procedure CROUT solves a system of linear equations A × y = b with different right hand sides. A is a square matrix of order n, y and b are vectors of dimension n. The problem is equivalent with solving a matrix equation A × Y = B when Y and B are matrices with n rows and m columns.

In order to deminish accumulated round-off errors Møller's device (ref[2]) is incorporated in the procedure. Please note that the procedure call differs slightly from the call of SA10.

2.Method.

The solution is based on Crouts method with row interchanges. First the matrix A is equilibrated, i.e. the row vectors are scaled with integers of the form 2^p where p is chosen so that a norm Ni for each row i satifies the inequality

$$1/2 \leq Ni \leq 1.$$

As a norm we use the maximum norm defined by

$$Ni = \max(\mathrm{abs}(A[i,j])),\ j = 1,2, \ldots\ldots\ldots n.$$

Then the matrix A is transformed into its triangular decomposition LU = A. L is a lower triangular matrix with all elements above the diagonal equal to zero and with elements L(k,k) = 1 for all k. U is an upper triangular matrix with all elements below the diagonal equal to zero. Next the right hand side b is transformed in accordance with the triangularization. When the triangularization has been finished the back substitution takes place giving the solution y corresponding to the b.

By scaling the rows in A to roughly the same maximum magnitude the procedure is allowed to select effective pivotal elements in the triangularization.

This algorithm is based on an original algorithm published in Com. of the ACM (ref[1]). The main differences are the introductions of scaling and Møller's device which increases the accuracy and the substitution of the real procedure INNERPRODUCT by for- statements resulting in a much faster algol procedure.

By the triangularization enough information will be stored for the procedure to transform a new column vector b without repeating the triangularization and we can then compute a new solution of the linear system with the same A but with a new right hand side.

The procedure CROUT also computes the determinant of the matrix A.

3. Use of the Procedure.

The procedure call shall be of the form

```
CROUT (n,A,b,f,pivot,det,repeat,exit);
```

The parameters are:

n : integer, is the order of the coefficient matrix

A : declared as real array A [1:n,1:n], holds the coeffient matrix of the equation system when CROUT is called first time (with repeat = false). Upon exit from CROUT, A holds the triangular matrices L and U and it must not be changed if subsequent calls with repeat = true are wanted.

b : declared as real array b [1:n], holds the right hand side before the call and the solution when the call is finished.

f : declared as ~~integer~~ array f [1:n] is assigned values which are factors originating and used in equilibration of the coefficient matrix A during the first call with repeat = false. These values must not be changed if subsequent calls with repeat = true are wanted.

pivot : declared as integer array pivot[1:n], is assigned values which are row indices originating from the triangularization of A during a call with repeat = false. These values must not be changed if subsequent calls with repeat = true are wanted.

det : real, is assigned the value of the determinant of A when repeat = false; otherwise it is unchanged in the call.

repeat : boolean, is false when CROUT is called first time with a coefficient matrix and true in subsequent calls with the same matrix.

exit : is a label to which the procedure exits if the matrix is singular. It must exist in the program in such a place that it is known at the position of the call.

4. References.

1. Communications of the ACM4, 1961, pp. 176-177.

2. O.Møller. Quasi Double-Precision in Floating Point Addition, BIT5(1965) pp. 37-50, and BIT5(1965) pp. 251-255.

O.Lang Rasmussen. K.Grau Sørensen.

5. Algorithm.

```
comment A.E.K. February, 1967 the procedure CROUT solves a
system of linear equations A × y = b with different b as described in SA10/1;
procedure CROUT(n,A,b,f,pivot,det,repeat,exit);
value n;
array A,b,f;
integer n;
integer array pivot;
real det;
boolean repeat;
label exit;
begin
integer k,i,j,imax,p;  real t,q, in,c,u,v,s;
if ¬repeat then
begin
det:=1;
for i:=1 step 1 until n do
begin comment equilibrate A[1:n,1:n];
q:=0; for j:=1 step 1 until n do
begin t:=abs(A[i,j]); if t>q then q:=t end;
if q=0 then go to exit;
f[i]:=t:=2↑(-entier(ln(q)/0.693147181+1));
for j:=1 step 1 until n do A[i,j]:=A[i,j]×t;
end equilibration;

for k:=1 step 1 until n do
begin comment triangularization;
t := 0; for i:=k step 1 until n do
begin u:=c:=0;
for p:=1 step 1 until k-1 do
begin v:=A[i,p]×A[p,k];
if abs(u)<abs(v) then begin in:=u; u :=v; v:=in end;
s:=u + v; c:=c + ((v-(s-u))+(u-(s-(s-u))));
u:=s
end;
in:=u + c;
q:=A[i,k] := A[i,k] - in;
if abs(q)>t then begin t:=abs(q); imax := i; end if;
end i;
pivot[k] := imax;
comment  the largest pivot element A[imax,k] in column k is found;
if imax ≠ k then
begin  comment interchange rows k and imax;
det := -det;
for j:=1 step 1 until n do
begin t :=A[k,j]; A[k,j] := A[imax,j]; A[imax,j] := t; end j;
end interchange;

q:= A[k,k];
if q = 0 then goto exit;
q :=1/q;
for i:=k+1 step 1 until n do
A[i,k] := q× A[i,k];
```

```
for j:=k+1 step 1 until n do
begin u:=c:=0;
for p:=1 step 1 until k-1 do
begin v:=A[k,p]×A[p,j];
if abs(u)<abs(v) then begin in:=u; u:=v; v:=in end;
s:=u + v; c:=c + ((v-(s-u))+(u-(s-(s-u))));
u:=s
end;
in:=u + c;A[k,j] := A[k,j] - in;
end j;
end k;
end -,repeat;
for i:= 1 step 1 until n do b[i] := b[i]×f[i];
for k:=1 step 1 until n do
begin comment process righthandside;
p:= pivot[k]; t := b[p]; b[p] := b[k]; b[k] := t;
u:=c:=0;
for p:=1 step 1 until k-1 do
begin v:=A[k,p]×b[p];
if abs(u)<abs(v) then begin in:=u; u:=v; v:=in end;
s:=u + v; c:=c + ((v-(s-u))+(u-(s-(s-u))));
u:=s
end;
in:=u + c;
b[k] := b[k] - in
end righthandside;

for k:=n step -1 until 1 do
begin  comment backsubstitution;
if -,repeat then det := A[k,k] × det/ f[k];
u:=c:=0;
for p:= k+1 step 1 until n do
begin v:=A[k,p]×b[p];
if abs(u)<abs(v) then begin in:=u; u:=v; v:=in end;
s:=u + v; c:=c + ((v-(s-u))+(u-(s-(s-u))));
u:=s
end;
in:=u + c;
b[k] := (b[k] -in)/A[k,k]
end backsubstitution;

end CROUT;
```

DANISH ATOMIC ENERGY COMMISSION,
Research Establishment Risø,
Reactor Physics Section,
Computer Group.

July 1962,
SA - 13,
25 copies.
Reprint 11/2-1966.

Polynomial Approximation.

ALGOL - procedure: POLY 1.

1. Scope.

The procedure calculates the coefficients a[o] .... a[P] in a polynomial approximation of a least squares fit to a function y = F(x) on the basis of N corresponding values of abscissa x and ordinates y eventually provided with weights w.

2. Method.

The coefficients in the polynomial are found by use of orthogonal polynomials as described in a paper by G.E. Forsythe in J.Soc.Indst. and Appl. Math. 5 (1957) p.p. 74 - 88.

3. Use of the procedure.

The procedure will be copied into the program where the following comment is written:

comment library POLY 1;

The procedure call is of the type:

POLY 1 (N,P,x,y,w,a,WEIGHING);

The parameters are:

N the number of (x,y,w) values, an integer

P the wanted polynomial degree, an integer

x a real array [1:N] in which the abscissa of the fitting points must be placed by the main program before call of the procedure.

y a real array [1:N] in which the ordinate of the fitting points must be placed by the main program before call of the procedure.

w a real array [1:N] in which the weights on the fitting points must be placed by the main program before call of the procedure if WEIGHING is false the w array can be the same as the y array

a a real array [0:P] in which the polynomial coefficients will be stored by the procedure

WEIGHING is a boolean, the value of which can be true, meaning use weights or false, meaning use no weights.

```
procedure POLY1(N,P,x,y,w,a,WEIGHING);
integer N,P;
boolean WEIGHING;
array x,y,w,a;

comment A.E.K. August 2nd.1962. Least squares fit of a polynomial approximation
as described in SA -13;

begin
integer j,k,n;
real alfa,beta,XPROD,YPROD,SQ,SQSUM,OLDSQSUM,R,olda;
array error, orpol,oldorpol[1:N], cora[-1:P], oldcora[0:P];

for n := 1 step 1 until N do
begin
error[n] := y[n];
orpol[n] := 0;
oldorpol[n] := 1
end of initial setting;

alfa := olda := cora[-1] := 0;
beta := OLDSQSUM := 1;

for k := 0 step 1 until P do
begin
XPROD := YPROD := SQSUM := 0;

for n := 1 step 1 until N do
begin
error [n] := error[n] - olda × orpol[n];
R := oldorpol[n] × beta;
oldorpol[n] := orpol[n]
R := orpol[n] := R + orpol[n] × (x[n] + alfa);
if WEIGHING then R := orpol[n] × w[n];
SQ := R × orpol[n];
SQSUM := SQSUM + SQ;
YPROD := YPROD + R × error[n];
XPROD := XPROD + SQ × x[n]
end for n;

a[k]:=olda:= YPROD/SQSUM;
oldcora [k]:= 0;
cora [k]:= 1;
if k>0 then

for j:= k-1 step -1 until 0 do
begin
R := beta × oldcora[j];
oldcora[j] := cora[j];
cora[j] := alfa × oldcora[j] + R + cora[j-1];
a[j] := a[j] + olda × cora[j]
end for j;

beta := -SQSUM/OLDSQSUM;
OLDSQSUM := SQSUM;
alfa := -XPROD/SQSUM
end for k;
end of POLY-1;
```

Erik Hansen.

DANISH ATOMIC ENERGY COMMISSION,
Research Establishment Risø,
Reactor Physics Section,
Computer Group.

July 1962.
SA-14.
25 copies.

Solution of a system of linear equations.

ALGOL - procedure: CROUT 2.

1. Scope.

This procedure which is a simplified version of SA-10 in that respect that it can solve only one system of linear equations Ay = b where A is a square matrix of order n, y and b columm vectors of dimension n. The procedure makes use of a real procedure INNERPRODUCT which is placed immediately before procedure CROUT 2.

2. Method.

The solution is based on Crouts method with row interchanges. First the matrix A is transformed into its triangular decomposition LU = A. L is a lower triangular matrix with all elements above the diagonal equal to zero and with elements L(k,k) = 1 for all k. U is an upper triangular matrix with all elements below the diagonal equal to zero. At the same time the right hand side b is transformed in accordance with the triangularization. When the triangularization has been finished the back substitution takes place giving the solution y corresponding to the b.
The procedure CROUT 2 also computes the determinant, det, of the matrix A.
Two nonlocal identifiers appears in the procedure

1. the nonlocal label ,singular, to which the procedure exits if matrix A is singular i.e. if the determinant of A is zero.

2. the real procedure INNERPRODUCT which forms a sum of products to be used in procedure CROUT 2. The real procedure INNERPRODUCT may be declared in the head of any block which includes the block in which procedure CROUT 2 is declared.

3. Use of the Procedure.

The procedure will be copied into the program where the following comment is written:

comment library CROUT 2;

The procedure call shall be of the form

CROUT 2 (A,b,n,y,det);

All parameters A,b,n,y,det have to be declared in the program before the call of procedure CROUT 2. A shall be declared as a real two-dimensional array A[1:n,1:n], b and y as one-dimensional arrays b,y[1:n], n is declared as an integer and det as a real.

Before the call of the procedure values must be assigned to A,b, and n(this ay be done by the programmer or by the program itself which uses the procedure).

By using this call the procedure CROUT 2 performs the triangularization of A, the transformation of b and solves the equations. The elements of the solution vector will by CROUT 2 be assigned to the real array y. The procedure will change the values of A and b but not n. The procedure will compute the determinant of A and assign the value to the real det. If the matrix A is singular an exit is made from the procedure CROUT 2 to the label singular in the program. At the place in the program to which the exit is wanted one writes, singular:followed by some statement, e.g. go to .... or other statement.

4. Comments to the procedure.

In this section some comments to the real procedure INNERPRODUCT and procedure CROUT 2 are written to facilitate the understanding of the procedures. Reference to these comments are made in the algorithm by numbers 4.1, 4.2 etc. referring to the following subsections. Reference should otherwise be made to Communications of the ACM 4, 1961, pp. 176-177.

4.1. INNERPRODUCT forms the sum of u(k) × v(k) for k = s, s+1, s+2,.... f-1, f. If s > f the value of INNERPRODUCT is zero. INNERPRODUCT may be declared in the head of any block which includes the block in which CROUT 2 is declared.

4.2. We have found that A[imax,k] is the largest pivot in columm k. Now we interchange the rows k and imax.

4.3 The row interchange is done. We proceed to the elimination.

4.4. The triangularization is now finished and we skip to the back substitution.

5. Algorithm.

```
comment A.E.K. july 16th. 1962 - the following procedure is a modified
version of SA-10 in that respect that it can solve a system of linear
equations Ay=b with only one b as described in SA-14 to which the following
comments refer;

real procedure INNERPRODUCT (u,v,k,s,f);
value s,f;
integer k,s,f;
real u,v;

comment section 4.1;

begin
real h;
h := 0;
for k := s step 1 until f do
h := h + u × v;
INNERPRODUCT := h
end INNERPRODUCT;

procedure CROUT 2(A,b,n,y,det);
value n;
array A,b,y;
integer n;
real det;
```

```
begin
integer k,i,j,imax,p;
real TEMP, quot;
det := 1;
for k := 1 step 1 until n do

L1 : begin
TEMP := 0;
for i := k step 1 until n do

L2: begin
A[i,k] := A[i,k] - INNERPRODUCT (A[i,p], A[p,k],p,1,k-1);
if abs(A[i,k]) > TEMP then

L3 : begin
TEMP := abs(A[i,k]); imax := i

end L3

end L2;

comment section 4.2;


if imax ≠ k then

L4 :begin det := - det;
for j := 1 step 1 until n do

L5 :begin
TEMP := A[k,j]; A[k,j] := A[imax,j]; A[imax,j] := TEMP

end L5;

TEMP := b[k]; b[k] := b[imax]; b[imax] := TEMP

end L4;

comment section 4.3;

if A[k,k] = 0 then go to singular;
quot := 1.0/A[k,k];
for i := k + 1 step 1 until n do
A[i,k] := quot × A[i,k];
for j := k + 1 step 1 until n do
A[k,j] := A[k,j] - INNERPRODUCT (A[k,p], A[p,j],p,1,k-1);
b[k] := b[k] - INNERPRODUCT (A[k,p],b[p],p,1,k-1)

end L1; go to L6;

comment section 4.4;

L6: for k := n step -1 until 1 do

L7: begin det := A[k,k] × det;
y[k] := (b[k] - INNERPRODUCT(A[k,p], y[p],p, k+1,n))/A[k,k]

end L7;

end CROUT 2;
```

DANISH ATOMIC ENERGY COMMISSION
Research Establishment Risø
Reactor Physics Department
Computer Group

April 1967
SA - 22/2
50 copies

Solution of a system of linear equations

ALGOL-procedure: DRUMCROUT 1

1. Scope.

The real procedure DRUMCROUT 1 written in GIER-ALGOL III solves a system of linear equations $A \times y = b$, where A is a square matrix of order n, y is the solution and b the right hand side, both are vectors of dimension n.

The procedure can solve a system with several right hand sides, b, without repeating the triangularization of matrix A, i.e. it can solve a matrix equation $A \times Y = B$ where A is a square matrix of order n, Y and B are matrices with n rows and m columns.

DRUMCROUT 1 assumes that the matrix A and a right hand side b are stored on drum. The solution y will be stored on drum, by the procedure.

2. Method.

The solution is based on Crouts method with pivoting as described in SA - 10/1 section 2, and will not be repeated here.

3. Use of the procedure

The determinant of the coefficient matrix is obtained as the value of DRUMCROUT 1 during the first call of the procedure with repeat = false. In calls with repeat = true, the value of DRUMCROUT 1 is undefined.

The procedure call shall be of the form:

```
DRUMCROUT1 (n,ta,tb,ty,tf,tp,tr,repeat,exit);
```

where the parameters are:

n: integer, the order of coefficient matrix;

ta: integer, the value of drumplace corresponding to the storage on the drum of the first row of the coefficient matrix A of dimension [1:n,1:n]. This matrix is stored compactly row by row but so that each row starts at the beginning of a drum track. When stored in this way the values of drumplace for two elements in the same columnn but on consecutive rows will differ by a quantity which is divisible by 40(= number of words on a drum track). The coefficient matrix A must be stored on the drum before the first call of the procedure. During the first call of the procedure (repeat = false) the elements of A are changed because A is triangulated. The triangulated matrix must not be changed if subsequent calls with new right hand sides (repeat = true) are wanted.

tb: integer, the value of drumplace corresponding to the storage on the drum of a real array of dimension [1:n] which holds the right hand sides used in turn.

ty: integer, the value of drumplace corresponding to the storage on the drum of a real array of dimension [1:n] which holds the solution vector y. It may be remarked that the value of ty can be set equal to the value of tb, in this case the right hand side will be replaced by the solution.

tf: integer, the value of drumplace corresponding to the storage on the drum of a real array of dimension [1:n] which holds the factors computed and used in equilibration of the coefficient matrix during the first call (repeat = false) of the procedure. These factors must not be changed because they are used in all subsequent calls (repeat = true) with the same coefficient matrix but with new right hand sides.

tp: integer, the value of drumplace corresponding to the storage on the drum of an integer array of dimension [1:n]. This array holds the pivots originating from the triangularization of the coefficient matrix during the first call (repeat = false) of the procedure. The pivots must not be changed because they are used in all subsequent calls (repeat = true) with the same matrix , but with new right hand sides.

tr: integer, the value of drumplace corresponding to the storage on the drum of an integer array of dimension [1:n]. This array holds the values of drumplace corresponding to the storage of the rows of the coefficient matrix A and they are stored during the equilibration process in the first call (repeat = false) of the procedure. These drumplace numbers are permutated during the triangularization according to the pivoting and must not be changed because they are used in all subsequent calls (repeat = true) of the procedure with the same matrix, but with new right hand sides.

repeat: boolean. During the first call of DRUMCROUT 1 this parameter must have the value false. In all subsequent calls with the same coefficient matrix but with new right hand sides it must have the value true.

exit: a label to which jump is made from DRUMCROUT 1 if the coefficient matrix is singular.

4. Running time and storage requirements.

The speed of computations depends of cause strongly on how the loop structure of the procedure matches with the tracks on the drum on which the procedure is stored. The procedure requires 21 drumtracks. Tests of the procedure with different systems have shown the following computation times:

order of system	computation time
20	40 sec
40	2 min 56 sec + 8 sec for every new right hand side
41	3 min 28 sec + 10 sec - - -
60	8 min 12 sec + 18 sec - - -
80	16 min 39 sec + 31 sec - - -
81	17 min 30 sec + 35 sec - - -

5. References.

1. Communications of the ACM, 4, 1961, pp 176-77.

2. SA-10/1 CROUT Algol procedure. February 1967

Ole Lang Rasmussen.

6. Algorithm.

```
comment A.E.K. April the 26th 1967. - the procedure DRUMCROUT 1 written
in GIER-ALGOL III solves a system of linear equations A × y = b with
different b as decribed in SA- 22/2;

real procedure DRUMCROUT1(n,ta,tb,ty,tf,tp,tr,repeat,exit);
value n,ta,tb,ty,tf,tp,tr;integer n,ta,tb,ty,tf,tp,tr;
boolean repeat;label exit;
begin
integer i,j,k,imax,gem,p,m,r,N;real t,q,h,det,detfactor;
real array a[1:n]; integer array pivot,row[1:n];
gem:=drumplace;
N:=(n:40)×40;if N<n then N:=N+40;
if-,repeat then
begin
det:=detfactor:=1;
k:=ta;
begin comment equilibration;
array a1[1:N];
for i:=1 step 1 until n do
begin
drumplace:=row[i]:=k;fromdrum(a1);h:=0;
for j:=1 step 1 until n do begin t:=abs(a1[j]);if t>h then h:=t end;
if h=0 then goto exit;
a[i]:=t:=2↑(-entier(ln(h)/0.693147181+1));
for j:=1 step 1 until n do a1[j]:=a1[j]×t;
detfactor:=detfactor×t;
drumplace:=k;k:=k+todrum(a1);
end i;
comment store factors on drum;
drumplace:=tf;todrum(a);
end equilibration;
for p:=40 step 40 until N do
begin comment triangularization;
m:=if p=40 then 0 else 1;
k:=p-40;
for k:=k+1 while k≤n∧k≤p do
begin
t:=0;
begin array a1[m:p-40],a2[p-39:p];
for i:=k step 1 until n do
begin
if p>40 then begin drumplace:=row[i]-N+p-40;fromdrum(a1) end;
drumplace:=row[i]-N+p; fromdrum(a2);
if k≠1 then
begin
if k-1≥p-39 then a2[k-1]:=a2[k-1]×q else a1[k-1]:=a1[k-1]×q;
h:=0;
for j:=1 step 1 until p-40 do h:=h+a1[j]×a[j];
for j:=p-39 step 1 until k-1 do h:=h+a2[j]×a[j];
h:=a2[k]:=a2[k]-h; drumplace:=row[i]-N+p; todrum(a2);
if k-1<p-39 then begin drumplace:=row[i]-N+p-40;todrum(a1) end;
end k≠1 else h:=a2[1];
if abs(h)>t then begin t:=abs(h);imax:=i end;
end i;
pivot[k]:=imax; comment largetst pivot element A[imax,k] is found;
if imax≠k then
```

```
begin comment interchange drumplace numbers;
det:=-det;i:=row[k];row[k]:=row[imax];row[imax]:=i;
end imax≠k;
end;
begin array a1[1:p];
drumplace:=row[k]-N+p;fromdrum(a1);
t:=a1[k];
if t=0 then goto exit;det:=det×t;
if k<n then
begin
r :=if k=p then p+1 else p-39;
begin array a2[k+1:n],a3[r:N];
q:=1/t;for i:=k+1 step 1 until n do a2[i]:=0;
for i:=1 step 1 until k-1 do
begin
drumplace:=row[i]; fromdrum(a3);a[i]:=a3[k+1];h:=a1[i];
for j:=k+1 step 1 until n do a2[j]:=a2[j]+h×a3[j];
end i;
drumplace:=row[k];fromdrum(a3);
for i:=k+1 step 1 until n do a3[i]:=a3[i]-a2[i]; a[k]:=a3[k+1];
drumplace:=row[k];todrum(a3);
end;
end k<n;
end;
end k;
end p,triangularization;
drumplace:=tr;todrum(row); drumplace:=tp;todrum(pivot);
DRUMCROUT1:=det/detfactor;
end-,repeat;
drumplace:=tb;fromdrum(a);
begin array f[1:n];
drumplace:=tf;fromdrum(f);
for i:=1 step 1 until n do a[i]:=a[i]×f[i];
end;
drumplace:=tp;fromdrum(pivot);
drumplace:=tr;fromdrum(row);
for p:=40 step 40 until N do
begin comment elimination of right hand side;
array a1[1:p];
k:=p-40;
for k:=k+1 while k<n∧k<p do
begin
t:=a[pivot[k]];a[pivot[k]]:=a[k];a[k]:=t;
drumplace:=row[k]-N+p;fromdrum(a1); h:=0;
for i:=1 step 1 until k-1 do h:=h+a1[i]×a[i]; a[k]:=a[k]-h;
end k;
end p,right hand side;
for p:=N-39 step -40 until 1 do
begin comment backsubstitution;
array a1[p:N];
m:=if p=N-39 then n else p+39;
for k:=m step -1 until p do
begin
drumplace:=row[k]; fromdrum(a1);h:=0;
for i:=k+1 step 1 until n do h:=h+a1[i]×a[i];
a[k]:=(a[k]-h)/a1[k];
end k;
end backsubstitution;
drumplace:=ty;todrum(a);drumplace:=gem;
endDRUMCROUT1;
```

DANISH ATOMIC ENERGY COMMISSION,
Research Establishment Risø,
Reactor Physics Section,
Computer Group.

June 13th 1963
SA-31/1
25 copies

Determination of a Zero of an Arbitrary Function
ALGOL-procedure : HYP.

1. Scope.

The following procedure determines with prescribed accuracy a zero of an arbitrary function in a prescribed interval. If the values of the function at the end points of the interval are different from zero and have the same sign, a jump to a prescribed label will occur.

2. Method.

The basic method is hyperbolic interpolation using two points with function values of the same sign and one point with function value of the opposite sign.

If this method would give a bad result, bisection is used, and after six successive failures the procedure will go over to bisection.

3. Use of the procedure.

The procedure will be copied into the programme where the following comment is written:

comment library HYP;

The procedure call must have the form:

HYP(x,F,x1,x2,eps,error);

x is the name of the zero

F is the name of a rel procedure (with one formal parameter) determining the function for which a zero is wanted.

x1 and x2 are the end points of the interval inside which a zero is to be determined.

eps is the accuracy with which one wants to determine x; eps is not specified as value, so that a relative accuracy e may be prescribed by inserting e × x (if x is the name of the zero) in the place of eps in the procedure call. e should not be chosen less than $5 \cdot 10^{-9}$, since otherwise round-off errors could increase the running time significantly.

error is the label mentioned in the first section.

4. Additional remark.

The difference between this HYP version and SA-31 is that the use of arrays is avoided, which diminishes the running time for the procedure itself. This may have some significance in cases where the F-procedure is not very slow.

5. Algorithm.

```
procedure HYP(x, F, x1, x2, eps, error);
value x1,x2; real x,x1,x2,eps; label error; real procedure F;

comment: A.E.K. - May 30th 1963 - this procedure locates
         a zero for the function F as described in SA-31/1;
begin
integer p; real x3,f1,f2,f3,T,N,f;
f1:=F(x1); f2:=F(x2); if f1×f2>0 then goto error; p:=0;
if f1=0 then x:=x1 else if f2=0 then x:=x2 else goto bis; goto out;

hyp:
T:= 1/f1-1/f3; N:=(1/f1-1/f2)/(x1-x2)+(1/f3-1/f2)/(x2-x3);
if N≠0∧abs(f3)>abs(f2) then x:=x+T/N else goto bis;
if abs(x-x2)>abs(x1-x2)/2 then
bis:
x:=(x1+x2)/2; f:=F(x); if f=0 then goto out;
if sign(f)=sign(f2) then
begin x3:=x2; f3:=f2; p:=p+1 end else
begin x3:=x1; f3:=f1; x1:=x2; f1:=f2; if p<6 then p:=0 end;
x2:=x; f2:=f;
if abs(x1-x2)>eps∧abs(x2-x3)>eps then
goto if p<6 then hyp else bis;
out:
end;
```

G.K. Kristiansen.

DANISH ATOMIC ENERGY COMMISSION,
Research Establishment Risø,
Reactor Physics Department,
Computer Group.

December 19th 1963
Reprint July 4th 1967
SA - 32/1
50 copies

Romberg Quadrature

ALGOL procedure: romberg 2.

1. Scope.

The procedure calculates a definite integral $\int_a^b F(x)dx$.

2. Method.

The integral is calculated by a method proposed by W. Romberg [1] and advocated by E. Stiefel, e.g. [2]. This algorithm is adapted from [3]. Round off errors are diminished according to [4]. This is the only difference between romberg and romberg 2.

If the integral is approximated by a sequence of polygones
T[0,0] = (b-a)/2×(f(a)+f(b)); T[0,1] = (b-a)/4×(f(a)+2×f((a+b)/2)+f(b)) etc.
each of which is found by the trapezoidal rule applied to subintervals, it may be shown that (T[0,K+1]-I)/(T[0,K]-I) → 1/4, for K → ∞
Here I is the correct value of the integral and it is supposed that F(x) may be expressed by a fourier-sum. A new sequence of values may now be formed, assuming that the value of I which makes this expression equal to 1/4 is a better approximation: T[1,K] = (4×T[0,K+1] - T[0,K])/3, which has the rate of convergence 1/16, and generally we find

$$T[m,K] = (4\uparrow m \times T[m-1,K+1] - T[m-1,K])/(4\uparrow m - 1);$$

The sequence T[m,0] converges towards the integral if F(x) is Riemannintegrable. It is calculated from 2↑(m+K)+1 values of F(x).

The calculation is finished when two successive T[m,0] agree to within a prescribed relative error, delta. Yet, this is not a sufficient criterion to ensure the correct value of the integral, so a further condition is introduced, equivalent to a minimum number of mesh points, which must be reached before the exit from the procedure may take place. On the other hand it may happen that the permitted error, delta, never is reached, so an upper number of mesh points is prescribed, upon which exit to an alarm label takes place. When each functionvalue is added to the sum, the difference between the addend and the increment in the sum is found, and this difference is summed separately, and finally added to the sum. In this way the round off errors are considerably deminished.

3. Use of the Procedure.

The procedure call is of the type:

```
a1 := b + romberg 2(F,x,a,b,delta,nmin,nmax,n,FORMANGE);
```

The parameters are

F — a real expression, which defines the function to be integrated. It must depend on the simple real variable

x — which is the integration variable. The value of x on exit from the procedure is not defined.

a — is the lower limit (real).

b — is the upper limit (real).

delta — is the relative tolerance on two successive approximations (real).

nmin — is the least number of subintervals permitted on exit (integer).

n — is the actual number of subintervals used by the procedure (integer). The call must hold an identifier, declared as integer, in the corresponding position.

nmax — is the maximum number of subintervals (integer). If n > nmax, exit takes place to the alarm label

FORMANGE

It should be noticed, that this call is identical with the call of SIMPSON 2 (SA-11). Yet for a certain n, SIMPSON2 has calculated 2n+1 values of F(x), while romberg2 only has found n+1 values. Tests indicate that romberg2 will find the answer in about half the time, the SIMPSON2 procedure needs.

4. Comments on two pit falls.

a. The reason why a comparison between two consecutive approximations is not sufficient to determine when the integration is finished will be seen from this example:

$$\int_0^{16\pi} \cos(x)\,dx$$

Number of subintervals	4	8
Romberg - value	16	16

but the true value is 0.

b. Consider the integral $\int_{0.2}^{4} 1/x\,dx$ It is evident that a much smaller mesh-size will be needed near the lower bound than near the upper one, so the calculation is performed much quicker if the integral is written as

$$\int_{0.2}^{1} 1/x\,dx + \int_{1}^{4} 1/x\,dx$$

5. Bibliography.

1. W. Romberg: Vereinfachte numerische Integration, Det Kong. Norske Videnskabers Selskab Forhandlinger, bind 28, nr. 7, Trondhjem 1955.
2. E. Stiefel: Altes und neues über numerische Quadratur. Zeitschrift für angewandte Mathematik und Mechanik (ZAMM),41,Heft 10-11,1961.
3. F.L.Bauer: Algorithm 60, ACM 4, nr.6, 1961.
4. O. Møller. Quasi Double - Precision in Floating Point Addition, BIT,5, (1965), pp 37-50 and BIT,5, (1965), pp 251-255.

6. Algorithm.

```
real procedure romberg2 (F,x,a,b,delta,nmin,nmax,n,FORMANGE);
value a,b,delta,nmin,nmax;
real F,x,a,b,delta; integer nmin,n,nmax; label FORMANGE;
begin real l,u,m,s,u1,err,f; integer g,h,j,ord; ord:=ln(nmax)×1.5;
begin real array t[0:ord];
l := b-a; x:=a; u := F; x := b; t[0] := (u+F)/2;
n := 2; s := 0; ord := ord-1;
for h := 0 step 1 until ord do
begin u := err := 0;
m := l/n;
for j := 1 step 2 until n-1 do begin
x := a+j×m; f := F; u1 := u + f; err := err +(f-(u1-u));
u := u1 end;
u := u1 + err;
b:=t[h+1] := u/n+t[h]/2;
g := 1;  for j := h step -1 until 0 do
begin g := 4×g;
b:=t[j] := b+(b-t[j])/(g-1)end;
if n≥nmin∧abs(s)×delta≥abs(b-s) then goto slut;
if n>nmax /2then goto FORMANGE ; n := 2×n; s := b  end;
slut : romberg2 := b×1 end end romberg2;
```

Leif Hansson.

DANISH ATOMIC ENERGY COMMISSION
Research Establishment Risø
Reactor Physics Department
Computer Group

June 1963
Reprint October 11th 1967
SA - 37
25 copies

Computation of the coefficients for a polynomial

ALGOL-procedure POLYCOEF.

1.Scope.

Let the polynomial

$P(z) = a[n]\times z\uparrow n + a[n-1]\times z\uparrow(n-1) + a[n-2]\times z\uparrow(n-2)+\ldots+a[1]\times z + a[0]$, where

$a[n]=1$, have the roots $m[1],m[2],\ldots\ldots,m[n]$. Then it can be written in the form

$P(z) = (z-m[1])\times(z-m[2])\times(z-m[3])\times\ldots\ldots\times(z-m[n-1])\times(z-m[n])$.

The procedure POLYCOEF computes the coefficients $a[n-1],a[n-2],\ldots,a[1],a[0]$ by means of the roots.

2. Method.

Let P(z) be a polynomial of degree p; by division of P(z) with z-m, we get a polynomial Q(z) of degree p-1 and a remainder r. The connection of a[p], a[p-1],....a[0] and the coefficients of Q(z), q[p-1],q[p-2]..... q[0] are given by the following equations:

$$
\begin{aligned}
q[p-1] &= a[p] \\
q[p-2] &= a[p-1] + m\times q[p-1] \\
q[p-3] &= a[p-2] + m\times q[p-2] \\
&\;\;\vdots \\
q[0] &= a[1] + m\times q[1] \\
r &= a[0] + m\times q[0]
\end{aligned}
$$

If m is a root of the equation P(z) = 0 then the remainder r is zero. Rearrangeing the equations we get

$$
\begin{aligned}
a[p] &= q[p-1] \\
a[p-1] &= q[p-2] - m\times q[p-1] \\
a[p-2] &= q[p-3] - m\times q[p-2] \\
&\;\;\vdots \\
a[1] &= q[0] - m\times q[1] \\
a[0] &= \;\; - m\times q[0]
\end{aligned}
$$

where r is put equal to zero because of the nature of the problem.

Let $q[p-1], q[p-2],\ldots\ldots q[0]$ where $q[p-1]=1$, be the coefficients for a polynomial Q(z) of degree p-1 corresponding to the roots $m[1]\ldots\ldots m[p-1]$, then with $m= m[p]$ the above equations give the coefficients $a[p]\ldots.a[0]$ where $a[p]=1$, for a polynomial P(z) of degree p corresponding to the roots $m[1]\ldots.m[p]$.

The method of computation therefore consists in repetetive use of the above equations with p varying from 1 to n.

3. Use of Algorithm.

It is assumed that both the roots, and the coefficients a[n],.....a[0] may be complex numbers.

The procedure call is of the type

```
POLYCOEF(n,A);
```

where

n is the degree of the Polynomial, declared as integer.

A is a one-dimensional array, declared as real array A[0:2×n+1]

Before the call of procedure POLYCOEF the roots of the polynomial must be stored in A, the real parts in elements of A with even indexes the imaginary parts in elements with odd indexes starting in A[2] and A[3] respectively.

The procedure stores the coefficients in A so that a[0] is stored in A[0],A[1], a[1]in A[2],A[3]...etc the real parts in elements of A with even indexes and imaginary parts in elements with odd indexes.

4. Algorithm.

```
procedure POLYCOEF(n,A);
value n;integer n;real array A;
begin comment this procedure computes the coefficients for a polynomial
      with given complex roots as described in SA-37;
integer i,j,m;
real r,s,x,y;
A[0]:=1; A[1]:=0; m:=2×n;
for i:= 2 step 2 until m do
begin r:=s:=0; x:= A[i]; y:= A[i+1];
for j:= i step -2 until 2 do
begin A[j] := A[j-2] - r×x+s×y;
      A[j+1]:= A[j-1]- s×x-r×y;
          r := A[j-2]; s:= A[j-1];
endj;
A[0] := - r×x + s×y;  A[1] := -s×x - r×y;
end i;
end procedure POLYCOEF;
```

O. Lang Rasmussen.

DANISH ATOMIC ENERGY COMMISSION
Research Establishment Risø
Reactor Physics Section
Computer Group

June 1963
SA - 38
25 copies

Transformation of variable in a complex polynomial.

ALGOL - procedure COMPOLYTRANS.

1. Scope.

Let P(z) = a[n]×z↑n+a[n-1]×z↑(n-1)+.....+a[1]×z+a[0] be a general complex polynomial; the procedure COMPOLYTRANS makes a transformation of the variable z given by z = c + f × z1 where z1 is the new variable, the factor f is a real constant and c a complex conThis transformation gives a new polynomial P1(z1) in z1.

2. Method.

The Taylor expansion of the polynomial P(z) around z=c is given by

$$P(z) = P(c) + (z-c)\times(d1P(c)/dz)/1! + (z-c)\uparrow 2\times(d2P(c)/dz\uparrow 2)/2! \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots + (z-c)\uparrow n\times(dnP(c)/dz\uparrow n)/n!$$

which can also be written as

$$P(z) = P(c) + ((z-c)/f)\times f\times(d1P(c)/dz)/1! + ((z-c)/f)\uparrow 2\times f\uparrow 2\times(d2P(c)/dz\uparrow 2)/2! \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots + ((z-c)/f)\uparrow n\times f\uparrow n\times(dnP(c)/dz\uparrow n)/n!$$

where d1P(c)/dz, d2P(c)/dz↑2, ..., dnP(c)/dz↑n are the first, second, ... and nth derivatives of P(z) for z=c. Substituting z=c+f×z1 we get

$$P1(z1) = P(c)+z1\times f\times(d1P(c)/dz)/1! + z1\uparrow 2\times f\uparrow 2\times(d2P(c)/dz\uparrow 2)/2! \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots + z1\uparrow n\times f\uparrow n\times(dnP(c)/dz\uparrow n)/n!$$

which is the Taylor expansion of P1(z1) around z1=0 expressed by the values of P(z) and its derivatives in z=c.

Division of P(z) by z-c gives the remainder P(c) and the quotient polynomial

$$(d1P(c)/dz)/1!+(z-c)\times(d2P(c)/dz\uparrow 2)/2!+\ldots\ldots+(z-c)\uparrow(n-1)\times(dnP(c)/dz\uparrow n)/n!$$

Division of this quotient polynomial by z-c gives the remainder (d1P(c)/dz)/1. and the quotient polynomial

$$(d2P(c)/dz\uparrow 2)/2!+\ldots\ldots+(z-c)\uparrow(n-2)\times(dnP(c)/dz\uparrow n)/n!$$

and so on.

When the remainders (d1P(c)/dz)/1., (d2P(c)/dz↑2)/2., (d3P(c)/dz↑3)/3., ... (dnP(c)/dz↑n)/n. are multiplied by f, f↑2, f↑3, ... f↑n we get the coefficients to z1, z1↑2, z1↑3, ... z1↑n in the expansion of P1(z1), i.e. the constant coefficients in the polynomial P1(z1). The computation of the coefficients in P1(z1) therefore consists in successive division of P(z) with z-c giving remainders which, except for the first, is multiplied by powers of f.

3. Use of the procedure.

The procedure call is of the type

COMPOLYTRANS(n,A,a,b,f);

n: degree of P(z), declared as integer.

A: one dimensional array containing the complex coefficients in P(z) declared as real array A[0:2 × n+1]. The real parts are stored in A[0], A[2], A[4], ... A[2xn] and the imaginary parts in A[1], A[3], A[5],.....A[2xn+1]; a[0] is stored in A[0], A[1], a[1] in A[2], A[3],.....a[n] in A[2xn], A[2xn+1] before the call of procedure.

a,b: are the real and imaginary parts of the constant c, they are declared as real.

f: a constant declared as real.

The procedure stores the coefficients of P1(z1) in A in the same order as those of P(z).

4. Algorithm.

```
procedure COMPOLYTRANS(n,A,a,b,f);
integer n;
real a,b,f;
real array A;
begin comment procedure COMPOLYTRANS makes a transformation of the variable
z in a complex polynomial P(z), such as described in SA-38;

integer i,j;
real s,t,u,v,q;
q:=1;
for i:=0 step 1 until n do
begin s:=t:=0;
      for j:=n step -1 until i do
      begin u:= A[2xj] + axs - bxt;
            v:= A[2xj+1] + axt + bxs;
            if j=i then begin u:=uxq; v:= vxq end;
            A[2xj]:=s:=u;  A[2xj+1] :=t:=v;
      end j;
      q:= qxf;
end i;
end COMPOLYTRANS;
```

O. Lang Rasmussen

ø

DANISH ATOMIC ENERGY COMMISSION
Research Establisment Risø
Reactor Physics Section
Computer Group

June 1963.
SA - 39
25 copies

Transformation of variable in a real polynomial.

ALGOL - procedure REPOLYTRANS.

1. Scope.

Let P(x) =a[n]xx↑n+a[n-1]xx↑(n-1)+.....+ a[1]xx+a[0] be a real polynomial; the procedure REPOLYTRANS makes a transformation of the real variable x given by x = a + f x x1 where x1 is the new variable, f and a are real constants. This transformation gives a new polynomial P1(x1) in x1.

2. Method.

The Taylor expansion of the polynomial P(x) around x=a is given by

$$P(x) = P(a) + (x-a)\times(d1P(a)/dx)/1! + (x-a)\uparrow 2\times(d2P(a)/dx\uparrow 2)/2! + \\ +\ldots\ldots+(x-a)\uparrow n\times(dnP(a)/dx\uparrow n)/n!$$

which can also be written as

$$P(x) = P(a)+((x-a)/f)\times f\times(d1P(a)/dx)/1! + ((x-a)/f)\uparrow 2\times f\uparrow 2\times(d2P(a)/dx\uparrow 2)/2! \\ +\ldots\ldots\ldots+((x-a)/f)\uparrow n\times f\uparrow n\times(dnP(a)/dx\uparrow n)/n!$$

where d1P(a)/dx, d2P(a)/dx↑2, ..., dnP(a)/dx↑n are the first, second, ... and the nth derivatives of P(x) for x=a. Substituting x=a+fxx1 we get

$$P1(x1) = P(a) + x1\times f\times(d1P(a)/dx)/1! + x1\uparrow 2\times f\uparrow 2\times(d2P(a)/dx\uparrow 2)/2! \\ +\ldots\ldots+ x1\uparrow n\times f\uparrow n\times(dnP(a)/dx\uparrow n)/n!$$

which is the Taylor expansion of P1(x1) around x1=0 expressed by the values of P(x) and its derivatives for x=a.

Devision of P(x) by x-a gives the remainder P(a) and the quotient polynomial

$$(d1P(a)/dx)/1! + (x-a)\times(d2P(a)/dx\uparrow 2)/2! +\ldots\ldots+(x-a)\uparrow(n-1)\times(dnP(a)/dx\uparrow n)/n!$$

Division of this quotient polynomial by x-a gives the remainder (dP1(a)/dx)/1. and the quotient polynomial

$$(d2P(a)/dx\uparrow 2)/2.+!\ldots\ldots+(x-a)\uparrow(n-2)\times(dnP(a)/dx\uparrow n)/n!$$

and so on.

When the remainders (d1P(a)/dx)/1., (d2P(a)/dx↑2)/2., (d3P(a)/dx↑3)/3....., (dnP(a)/dx↑n)/n. are multiplied with f, f↑2, f↑3, ...f↑n we get the coefficients to x1, x1↑2, x1↑3, ...x1↑n in the expansion of P1(x1), i.e. the constant coefficients in the polynomial P1(x1). The computation of the coefficients in P1(x1) therefore consists in successive division of P(x) with x-a giving remainders which, except for the first is multiplied by powers of f.

3. Use of procedure.

The procedure call is of the type

REPOLYTRANS(n, A, a, f);

n: degree of P(x) declared as integer;
A: a one dimensional array containing the coefficient in P(x); declared as real array A[0:n]; a[0] is stored in A[0], a[1] in A[1], ... a[n] in A[n] before the call of procedure.
a, f: are constants declared as real.

The procedure stores the coefficients of P1(x1) in A in the same order as those of P(x).

4. Algorithm.

```
procedure REPOLYTRANS (n, A, a, f);
value n, a, f;
integer n; real a, f;
real array A;
begin comment procedure REPOLYTRANS makes a transformation of the variable
x, in a real polynomial P(x), such as described in SA-39;
integer i, j;
real s, u, q;
q := 1;
for i := 0 step 1 until n do
begin s := 0;
for j := n step - 1 until i do
begin u := A[j] + a × s; A[j] := s := u end j;
A[i] := u × q; q := q × f;
end i;
end REPOLYTRANS;
```

O. Lang Rasmussen.

DANISH ATOMIC ENERGY COMMISSION
Research Establishment Risø
Rector Physics Seksion
Computer Group

September 16th 1963.
SA - 42
25 copies

Determination of the maximum of a function with one and only one maximum.

1. Scope.

The following procedure determines the maximum value of a function with one and only one maximum in a prescribed interval.

2. Method.

The method is based upon a division of the interval into 4 smaller intervals and the procedure determines within which of these the maximum must be. These intervals are then again divided into 4 intervals and so on until the intervals are smaller than delta. The use of arrays makes the procedure usefull in cases where the function is slow.

3. Use of the procedure.

The procedure will be copied into the pogram where the following comment is written.

comment library MAX;
The procedure call must be of the form:

MAX (F,x,a,b,delta);

Where the parameters are:

F is the name of the function for which a maximum value is wanted, and F is a real expression.

x is the variable. On exit from the procedure the value of x is not defined.

a,b are the end points of the interval where the function is considered, and a is the lower limit.

delta is the accuracy with which one wants to determine the argument for the maximum value;delta is not specified as a value; so that a relative accuracy ''e'' may be prescribed by inserting ''exx'' in the place of delta in the procedure call.

As trial functions were used

F1 = 3,5xcos(x)	a = -1; b = 3;
F2 = 3,5xcos(x)	a = 0; b = 3;
F3 = ln(x) - x	a = 0,5; b = 3;
F4 = -x/2 + 4xx + 2	a = 0; b = 4;

F5 = -3,675xx↑8 + 0,003xx↑2 + 3,141592 a = 0; b = 10;

F6 = -0,73146xx↑2 + 4,31467xx + 7,141952 a = -15; b = 35;

The accuracy required was delta = 10-6.

The members of calculations of functions were respectively :48, 25, 48, 49, 46, 51.

The calculations of the maximum of the above functions were per.functions about 10 sec.

4. Algorithm.

```
real procedure MAX(F,x,a,b,delta);
value a,b; real F,x, a,b,delta;

begin
real o,p,y;integer j; real array r[0:4];
o:= a; p:= b;
A: y:= (p-o)/4; x:=o; r[0]:= F; j:= 0;
B: j:= j+1; if j= 5 then
begin if (p-o)<delta then
begin
MAX:=r[4]; goto stop;
end;
o:=o+3xy; goto A;
end;
x:=o+jxy; r[j]:= F; if r[j]>r[j-1] then goto B;
D: if j=1 then
begin
if (p-o)<delta then begin MAX := r[0];goto stop; end;
p:=o+y; y:= (p-o)/4; j:= 0; goto B;
end;
E: if (p-o)<delta then
begin MAX := r[j-1]; goto stop; end;
o:= o+(j-2)xy; p:= o+2xy;
y:= (p-o)/4 ; r[0]:=r[j-2]; r[4]:=r[j]; r[2]:= r[j-1];
x:=o+y; r[1]:=F; x:=o+3xy; r[3]:=F; j:=1;
if r[j] < r[j-1] then goto D; j:= j+1;
if r[j] < r[j-1] then goto E; j:= j+1;
if r[j] < r[j-1] then goto E; j:= j+1;
if r[j] < r[j-1] then goto E; goto B;
stop: end;
```

K.Møller Pedersen.

DANISH ATOMIC ENERGY COMMISSION
Research Establishment, Risø
Reactor Physics Section
Computer Group.

December 1963
SA - 49
25 copies

Computation of the characteristic polynomial of a matrix

ALGOL - procedure Danilevski

1. Scope.

By means of the method of A.M Danilevski the coefficients of the characteristic polynomial of a real matrix A is found.

2. Method.

The given matrix A is transformed to its companion matrix C by applying a finite sequence of similarity transformations

$$A(k+1) = S(k)^{-1} A(k) S(k), \quad k = 0,1,2.....$$

where $A(0) = A$. $S^{-1}(k) S(k) = E$; E = unit matrix.

A and C are similar matrices and therefore they have the same characteristic equation. The companion matrix C of A is of the form

$$C = \begin{vmatrix} 0 & 0 & 0 & 0 & c[1,n] \\ 1 & 0 & 0 & 0 & c[2,n] \\ 0 & 1 & 0 & 0 & c[3,n] \\ - & - & - & - & - \\ - & - & - & - & - \\ - & - & - & - & - \\ 0 & 0 & 0 & 1 & c[n,n] \end{vmatrix}$$

and the characteristic polynomial of C is

$$z\uparrow n - c[n,n] \times z\uparrow (n-1) - c[n-1,n] \times z\uparrow (n-2) - \ldots\ldots - c[2,n] \times z - c[1,n] = 0$$

which is also the characteristic polynomial of A because of the similarity of A and C.

The transformation of A to C is performed such that A is first transformed to almost triangular form (Hessenberg form) i.e.a matrix H of the form

$$H = \begin{vmatrix} h[1,1], & h[1,2], \ldots\ldots\ldots h[1,n] \\ 1 \quad , & h[2\ 2], \ldots\ldots\ldots h[2,n] \\ 0 \quad , & 1 \quad , \ldots\ldots\ldots h[3,n] \\ - \quad , & - \quad , \ldots\ldots\ldots - \\ 0 \quad , & 0 \quad , \ldots\ldots\ldots 1, h[n,n] \end{vmatrix}$$

During this sequence of transformations we employ a search for pivots as is advocated for in [1]. The similarity transformations are not executed by elementary matrices as in [1] but follows the line of Faddeeva [2]. Secondly the almost triangular form is transformed to C.

We first describe the transformation to almost triangular form. We begin by zeroing the elements a[n,j] of A for j = 1,2,....n-2. This is done by postmultiplication of A by the matrix.

$$L(0) = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ - & - & - & - \\ - & - & - & - \\ - & - & - & - \\ -\frac{a[n,1]}{a[n,n-1]} & -\frac{a[n,2]}{a[n,n-1]}, & \frac{1}{a[n,n-1]}, & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

where $a[n,1], a[n,2], \ldots . a[n,n-1]$ are elements of $A(0)$.

The element $a[n,n-1]$ is the pivotal element for this operation and as we want it to be relatively large if round-off errors shall be as small as possible we make a search for the greatest element among $a[n,j]$ for $j = 1,2, \ldots . n-1$ followed by an exchange of respective columns. An exchange of columns is equivalent with a postmultiplication of an elementary matrix, and to secure the similarity this must be followed by an exchange of corresponding rows which is equivalent to premultiplication with the inverse of that elementary matrix. After the largest (in magnitude) off-diagonal element of A is moved to the position $(n,n-1)$ we know that

$$\text{abs}(a[n,j]/a[n,n-1]) \leq 1$$

for $1 \leq j \leq n-2$ and hence the operation to zero of $a[n,j]$ for these values of j is a relatively accurate proces. It shall be remarked that the elements $a[i,n]$ for $i = 1,2, \ldots . n$ are not affected by this operation because of the form of $L(0)$. This postmultiplication of A by $L(0)$ shall be followed by a preultiplication of $A(0)L(0)$ with the inverse $L^{-1}(0)$. It is easily verified that

$$L^{-1}(0) = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ - & - & - & \\ - & - & - & \\ a[n,1], & a[n,2], & , a[n,n-1], & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

as $L(0)\ L^{-1}(0) = E$. (E = unit matrix). When this operation is finished we have a matrix

$$A(1) = L^{-1}(0)\ A(0)\ L(0)$$

which is similar to $A(0)$ and which elements in the positions $(n,1),(n,2) \ldots . .$ $(n,n-2)$ are zero while the element in positions $(n,n-1)$ is 1.

The next step is to zero the elements in $A(1)$ in the positions $(n-1,1)$, $(n-1,2), \ldots . (n-1,n-3)$, while the element in position $(n-1,n-2)$ shall be 1. This proces is done by the similarity transformation $A(2) = L^{-1}(1)A(1)L(1)$ where

$$L(1) = \begin{vmatrix} 1 & 0 & 0 & 0 , 0 \\ 0 & 1 & 0 & 0 , 0 \\ - & - & - & - \ - \\ - & - & - & - \ - \\ - & - & - & - \ - \\ -\frac{a[n-1,1]}{a[n-1,n-2]}, & -\frac{a[n-1,2]}{a[n-1,n-2]}, & \frac{1}{a[n-1,n-2]}, & 0 , 0 \\ 0 & 0 & 0 & 1 , 0 \\ 0 & 0 & 0 & 0 , 1 \end{vmatrix}$$

and

$$L^{-1}(1) = \begin{vmatrix} 1 & 0 & 0 & 0,0 \\ 0 & 1 & 0 & 0,0 \\ - & - & - & - \\ - & - & - & - \\ a[n-1,1], & a[n-1,2], & ,a[n-1,n-2], & 0,0 \\ 0 & 0 & 0 & 1,0 \\ 0 & 0 & 0 & 0,1 \end{vmatrix}$$

Here a[n-1,1],a[n-1,2],....a[n-1,n-2] are elements in the matrix A(1). Again an exchange of columns and rows must preceed the post- and premultiplication with L(1) and $L^{-1}(1)$ if the pivotal element now in position (n-1,n-2) is not the largest in magnitude.

Continuing these processes we finally arrive to the almost triangular form H.

In the second part of the transformation i.e. the transformation from almost triangular form H to the companion matrix C we wish to zero elements on and above the diagonal.

We begin with zeroing the element in the first column of H. This is obtained by premultiplication of H with a matrix of the form

$$U(0) = \begin{vmatrix} 1, & -h[1,1], & 0, & 0 \\ 0, & 1, & 0, & 0 \\ - & - & - & - \\ 0, & 0, & 0, & 1 \end{vmatrix}$$

followed by a post multiplication by $U^{-1}(0)$, the inverse of U(0). It is easily verified, that

$$U(0)^{-1} = \begin{vmatrix} 1, & h[1,1], & 0, & 0 \\ 0, & 1, & 0, & 0 \\ - & - & - & - \\ 0, & 0, & 0, & 1 \end{vmatrix}$$

because $U(0)U(0)^{-1} = E$. When the operation is finished we have the matrix

$$H(1) = U(0)HU^{-1}(0)$$

which is similar to H.

The next step is to zero the appropriate elements in the second column. This is done by the similarity transformation

$$H(2) = U(1)H(1)U(1)^{-1}$$

where

$$U(1) = \begin{vmatrix} 1, & 0, & -h[1,2], & 0, & , & 0 \\ 0, & 1, & -h[2,2], & 0, & , & 0 \\ 0, & 0, & 1, & 0, & , & 0 \\ - & - & - & - & & - \\ 0, & 0, & 0, & 0, & , & 1 \end{vmatrix}$$

and

$$U(1)^{-1} = \begin{vmatrix} 1, & 0, & h[1,2], & 0, & , & 0 \\ 0, & 1, & h[2,2], & 0, & , & 0 \\ - & - & - & - & & - \\ - & - & - & - & & - \\ 0, & 0, & 0, & 0, & , & 1 \end{vmatrix}$$

h[1,2] and h[2,2] are now elements in H(1).

Continuing this proces we finally arrive to the companion matrix C of A. If a subdiagonal) pivot element is found to be zero in applying these processes, that is, if at an intermediate stage of the method we obtain a matrix of the form

$$A(k) = \begin{vmatrix} B & C \\ O & D \end{vmatrix}$$

for some k where B and D are square blocks and O zero matrix, we cannot zero the appropriate elements in the last column of B. But in this case the matrix partitions and we can therefore apply the method to B and D separately.

3. Use of procedure.

The procedure call is of the type

Danilevski (n,A);

where

n, declared as integer is the order of the matrix

A, declared as real array A[1:n,1:n] contains the elements of the given matrix which must be stored before the procedure call. The companion matrix is stored by the procedure in array A and the last column of the companion matrix stored in the array elements

a[1,n], a[2,n], ... a[n,n]

are the coefficients in the characteristic equation

$$z\uparrow n - a[n,n]\times z\uparrow(n-1) - a[n-1]\times z\uparrow(n-2) - \ldots\ldots\ldots - a[2,n]\times z - a[1,n] = 0$$

of the companion matrix as well as the original matrix.

4. References.

[1]: On the Danilevski method by Eldon R. Hansen
J.A.C.M. 10(1963) p. 102-109.

[2]: Computational Methods of Linear Algebra by
V.N. Faddeeva 1959.

5. Algorithm.

```
procedure Danilevski(n,A);
value n;integer n;real array A;
begin comment this procedure computes the coefficients of the characteristic
polynomial of a matrix with real elements such as described in SA-49;
integer i,j,k,k1;
real temp;
comment transformation of matrix to companion matrix;
if n=1 then goto L3;
for i:=n step -1 until 2 do
begin comment transform A to almost triangular form;
comment find greatest pivot element in row i;
temp:=0;
for k:=1 step 1 until i-1 do
if abs(A[i,k])>temp then begin temp:=abs(A[i,k]);k1:=k end;
if temp=0 then goto L1;
comment exchange columns and rows in A;
for j:=1 step 1 until i do
begin temp:=A[j,k1]; A[j,k1]:= A[j,i-1]; A[j,i-1]:=temp end j;
for k:=1 step 1 until n do
begin temp:= A[k1,k]; A[k1,k]:= A[i-1,k]; A[i-1,k]:=temp end k;
for k:=1 step 1 until i-2 do
begin comment arrayA[1:i-1,1:i-2] is changed by postmultiplication;
temp:=A[i,k]/A[i,i-1];
for j:=1 step 1 until i-1 do A[j,k]:= A[j,k]-A[j,i-1]×temp;
end k;
temp:=1/A[i,i-1];
for j:=1 step 1 until i-1 do A[j,i-1]:= A[j,i-1]×temp;
for k:=1 step 1 until n do
begin comment arrayA[1:i-1,1:i-2] is changed by premultiplication;
temp:=0;
for j:=1 step 1 until i-1 do temp:=temp+A[i,j]×A[j,k];
A[i-1,k]:=temp;
end k;
comment change row i in A;
for k:=1 step 1 until i-2 do A[i,k]:=0; A[i,i-1]:=1;
L1:end end transform to almost triangular form;
k1:=1;
for k:=1 step 1 until n-1 do
begin comment transform almost triangular form to companion matrix;
if A[k+1,k]=0 then begin k1:=k+1; goto L2 end;
for i:=k1 step 1 until k do
begin temp:=A[i,k];
for j:= k+1 step 1 until n do A[i,j]:= A[i,j]-temp×A[k+1,j] end i;
for i:= k1 step 1 until k do A[i+1,k+1]:=A[i+1,k+1]+A[i,k] ;
 or i:=k1 step 1 until k do A[i,k]:=0;
L2:end k;

L3:end Danilevski;
```

O. Lang Rasmussen.

DANISH ATOMIC ENERGY COMMISSION
Research Establishment Risø
Reactor Physics Section
Computer Group.

December 1963
SA - 51
25 copies.

Random numbers

ALGOL procedure: random.

1. Scope.

"random" generates a sequence of rectangularly distributed, pseudo-random numbers, one in each call of the procedure.

2. Method.

The successive pseudo-random numbers are generated from the formula

random [n+1] := 125 × random[n] (mod 2796203).

This formula generates a permutation of all integers $0 < i < 2796203$. Different values af random [0] cause only a cyclic shift in the generated numbers. The less significant bits should not be expected to be random.

Most litterature on this subject is concerned with the production of random digits in all positions and with taking advantage of the structure of the computer and the use of overflow. No such considerations are relevant to an algolprocedure, on the opposite overflow must be avoided.

Each call of the procedure takes approximately 8.6ms. Additive generators are faster, but are reported not to be statistical satisfactory. The advantage in speed is however less than what may be gained by rewriting the procedure into the program so that there are no formal parameters. Also the placing of the procedure across a track transition is of equal importance. Consequently additive generators have not been considered further.

3. Use of the procedure.

The procedure call is of the type

```
a := b + random (A,B,x0);
```

Here A and B are the bounderies of the interval in which the random numbers are distributed. $A < \text{random} < B$.

In the very first call, x0 must be used as the value of random [0]. This may be done by using a 6-digit number. In subsequent calls, x0 = 0.

4. Algorithm.

```
real procedure random (A,B,Y);
value A,B,Y; real A,B; integer Y;
begin integer C;
own integer X;
if Y≠0 then X:=Y;
C:=X×125;
X:=C-2796203×entier(C/2796203);
random:=X×(B-A)/2796203+A end random;
```

5. Tests.

The randomness of the numbers have been tested in several ways. N is the number of calls in each test. All tests use the first N numbers with the starting value x0 = 100001.

N	$1/N \sum r$	$1/N \sum (r \uparrow 2)$
50	0.50235	0.32714
100	0.48140	0.30822
500	0.49729	0.32628
1000	0.49701	0.32852
100000	0.50059	0.33363
200000	0.50018	0.33327
Expectance value	0.5	0.33333

10.000 and 90.000 random numbers were formed. The interval was divided into 100 equal parts and the number (fi) of random numbers within each part was counted.

chi-square = k/N × sum((fi-N/k) ↑ 2) was computed. k = 100 intervals.

For N = 10000, chisquare was 112.46, which corresponds to a probability of 80 - 90 percent with 99 degrees of freedom.

For N = 90000 the value was 98.53, which corresponds to a probability of 50 - 60 percent.

An autocorrelation of 3000 random numbers with shifts up to 511 gave 5.9 percent as the biggest correlation coefficient.

The period is as mentioned above 2796202 different numbers.

6. Reference.

Hull, T. E. and Dobell, A. R.:
Random Number Generators, SIAM Review, 4, 3, July 1962, pp.230-254, with 148 references.

Leif Hansson.

DANISH ATOMIC ENERGY COMMISSION
Research Establishment Risø
Reactor Physics Section
Computer Group.

July 8th 1964
SA - 64
25 copies

Solution of a system of linear equations

ALGOL - procedure CROUT 3.

1. Scope.

The real procedure CROUT 3 solves a system of linear equations Ay = b where A is a square matrix of order n, y and b are column vectors of dimension n. The value of CROUT 3 is the determinant of the coefficient matrix A. The difference between CROUT 3 and CROUT 2 is that the central part (the innerproduct) of CROUT 3 is written in machine code which has the effect that the speed of computation is increased in comparison with CROUT 2.

2. Method.

The solution is based on Crouts method with row interchanges as described in SA-14/1 and is therefore not repeated here.

CROUT 3 and CROUT 2 are completely equivalent, the only difference is that the innerproduct is written in code in stead of the for-statements in CROUT 2 (and the algol procedure INNERPRODUCT in former issues) which results in a considerably increased speed of computation.

3. Use of procedure.

The procedure consists of two parts, the algol procedure, and a code. The procedure call shall be of the form

CROUT 3(n,A,drum,length,exit);

where the parameters are:

n: declared as integer is the number of equations

A: declared as real array A[1:n,1:n+1] contains on entry the coefficient matrix in A[1:n,1:n] and the right hand side in the column A[1:n,n+1]. On exit the solution vector is stored in the column vector A[1:n,n+1].

drum: declared as integer contains the value of the standard variable drumplace, when the innerproduct in code is stored on the drum by the standard procedure gierdrum(see below).

length: declared as integer contains the number of machine words occupied by the code as supplied by the standard procedure gierdrum (see below).

exit: a label to which CROUT 3 goes when the matrix is singular.

The innerproduct in code must be stored on the drum before the call of CROUT 3. This shall be done by the program and can be performed in the following manner.

Assuming that the variable drum has been assigned some value of drumplace, one writes e.g. in the beginning of the program:

```
if kbon then
begin
writetext(‡<
message to operator: innerproduct-code in tapereader‡);
typechar;
drumplace:=drum;
gierdrum(‡<innerp‡,length);
writetext(‡<
message to operator: datatape in tapereader‡);
typechar;
end;
```

When the program is started after the translation is completed GIER stops ready for input of code. Started again it stops ready for input of data after which the computation starts. The first parameter, ‡<innerp‡, in the procedure call gierdrum (‡<innerp‡,length) is a code identification for innerproduct, which must be written exactly in this way. The second parameter, length, is assigned the number of machine-words in the code.

The procedure identifier CROUT 3 contains on exit the value of the determinant. It shall be remarked that all elements in A[1:n,1:n+1] are changed by the procedure.

4. References.

1. Communications of the ACM, 4,1961,pp.176-77.
2. SA-14/1, July 1964.

5. Algorithm.

```
comment A.E.K. this algol procedure solves a system of linear equation
as described in SA 64. The algorithm uses innerproduct in code;

real procedure CROUT 3(n,A,drum,length,exit);
value n,drum,length;integer n,drum,length;
real array A;
label exit;
begin
integer i,j,imax,k,p1,p2,p3,p4,gem;
real t,q,det,detfactor;
boolean array code[1:length];
boolean entry;
gem:=drumplace;
drumplace:=drum;
fromdrum(code);
gierproc(code[2],A,p1,p2,p3,p4,entry);
detfactor:= det := 1;
for i:= 1 step 1 until n do
begin comment equilibrateA[1:n,1:n+1];
q:= 0;
for j:= 1 step 1 until n do
begin t:= abs(A[i,j]); if t > q then q:= t end;

if q= 0 then goto exit;
t:= 2 ↑ (-entier(ln(q)/0.693147181+1));
for j:= 1 step 1 until n+1 do A[i,j]:= A[i,j]xt;
detfactor:= detfactorxt;
end equilibration;
p3:=1;
for k := 1 step 1 until n do
begin comment triangularization starts;
```

```
t := 0;
p2:=k;p4:=k-1;
for i := k step 1 until n do
begin
p1:=i;
A[i,k] := A[i,k] - gier(entry);
if abs(A[i,k]) > t then
begin t := abs(A[i,k]); imax := i end;
end;
comment the largest pivot element A[imax,k] in column k is found;
if imax ≠ k then
begin comment interchange rows k and imax;
det := -det;
for j := 1 step 1 until n+1 do
begin t := A[k,j]; A[k,j] := A[imax,j]; A[imax,j] := t end;
end interchange of rows;
if A[k,k] = 0 then goto exit;
q := 1/A[k,k];
for i := k + 1 step 1 until n do
A[i,k] := q × A[i,k];
p1:=k;
for j := k + 1 step 1 until n+1 do
begin
p2:=j;
A[k,j] := A[k,j] - gier(entry)
end;
end triangularization;
p2:=n+1; p4:=n;
for k := n step -1 until 1 do
begin comment backsubstitution;
p1:=k; p3:=k+1;
det := A[k,k] × det;
A[k,n+1] :=(A[k,n+1] - gier(entry))/A[k,k];
end backsubstitution;
CROUT 3:= det/detfactor;
drumplace:=gem;
end CROUT 3;
```

6. Innerproduct in code.

```
[innerproduct in code]
b a12
m
qq 10.3+39.9+41.15+53.21+37.27+37.33+57.39;‡<innerp>
r
[first entry]
        arn a3      D           ;Radr:=jump adress
        ar a1       ,gr(p9)     ;entry:=hv-instruction
        arn p4      ,gr a1      ;array description
        arn(a1)     ,tkfm1      ;Radr:=c2=n+2
        ga a4       ,tkm-10     ;a4[adr]:=c2
        gt a8                   ;a8[tæl]:=c2
        arn(a1)     t-1         ;
        ar (a1)     t-1         ;
        tkm 30      ,gr a1      ;a1[adr]:=array length+constant
        arn p4      ,tkm 10     ;Radr:=adr[last element]+1
```

```
            sr a1       ,gr a1     ;a1[adr]:=basisadr-constant
            it(p5)      ,pa a3     ;a3[adr]:=p1
            it(p6)      ,pa a10    ;a10[adr]:=p2
            it(p7)      ,pa a11    ;a11[adr]:=p3
            it(p8)      ,pa a12    ;a12[adr]:=p4
            hr s1                  ;return to ALGOL

[later entries]

a3:         arfnm0      ,tkfm1     ;Radr:=p1
            xr                     ;Madr:=p1
a4:         mknm0       D          ;R18:=p1xc2
            tkm9        ,ar a1     ;Radr:=p1xc2+basisadr-constant
            ga a7                  ;a7[adr]:=Radr
a10:        arfnm0      ,tkfm1     ;Radr:=p2
            ar a1       ,ga a8     ;a8[adr]:=basisadr+p2-constant
a11:        arfnm0      ,tkfm1     ;
            ga a2                  ;a2[adr]:=p3
            srm1        DX         ;Radr:=p3-1,Madr:=Radr
            mkn(a4)     D          ;R18:=(p3-1)xc2
            tkm9        ,ac a8     ;a8[adr]:=basisadr+p2+(p3-1)xc2-constant
a12:        arfnm0      ,tkfm-9    ;Rtæl:=p4
            gt a5       ,gp a6     ;a5[tæl]:=p4,a6[adr]:=p
a2:         ppm0        ,grn a9    ;p:=p3,a9:=0
a5:         bs p        t0         ;if p>p4 then a6 else a7
a6:         ppm0        ,hr s1     ;return
a7:         arfn p0                ;RF:=A[p1,p]
a8:         mkfm0       t0         ;RF:=A[p1,p]xA[p,p2]
            arf a9      ,grf a9    ;accumulate product
            pp p+1      ,hv a5     ;p:=p+1,jump
a1:         hv                     ;hv-instruction,storage for basisadr-constant
a9:         qq                     ;working cell
e
s
```

O. Lang Rasmussen.

DANISH ATOMIC ENERGY COMMISSION
Research Establishment Risø
Reactor Physics Section
Theoretical Dynamics Group

September 1st 1964
SA - 65
25 copies.

Description of ALGOL - procedure SA - 65

COMPFIT

Scope.

This procedure deals with the problem of fitting a known complex analytical function of a complex variable with a rational algebraic function of this complex variable.

The known function may be given by experimental or computed values of real and imaginary part, corresponding to a series of purely imaginary values of the argument.

Method.

The method used is originally developed by E. C. Levy (ref. 1) and later improved by C. K. Sanathanan and J. Koerner (ref. 2). The present version is that of ref. 2, slightly modified by the author.

In the following a very short description of the method of ref. 2:

The known function $F(s)$ is given in the m points $sk = j\omega k$, $k = 1,2,\ldots,m$ by the quantities $Rk(j\omega k)$ (real part) and $Ik(j\omega k)$ (imaginary part). The rational algebraic function is $G(s) = P(s)/Q(s)$, where $P(s)$ and $Q(s)$ are polynomials.

The difference between the complex values of $F(j\omega k)$ and $G(j\omega k)$ is

$$\varepsilon k = F(j\omega k) - P(j\omega k)/Q(j\omega k)$$

The criterion for the best possible fit as expressed by the least squares method would be: $\sum_{k=1}^{m} |\varepsilon k|^2$ adjusted to minimum.

However, this criterion leads to a system of very complex, non-linear equations and is consequently substituted by the following as a first approximation:

$$\sum_{k=1}^{m} |\varepsilon k \; Q(j\omega k)|^2 \text{ adjusted to minimum.}$$

This criterion leads to a matrix equation:

$$[A]\,[X] = [B] \qquad \text{(see ref. 1)}$$

there $[A]$ is a quadratic matrix of order $p+q+1$, p and q being the degrees of the polynomials $P(s)$ and $Q(s)$ respectively, while $[X]$ is a vector presenting the desired coefficients of these polynomials, assuming the zero-order term of $Q(s)$ adjusted to unity.

Now, denoting the fit function polynomials which results from this first approximation by P'(s) and Q'(s) a second approximation is performed according to the criterion

$$\sum_{k=1}^{m} \frac{|\mathcal{E}_k Q'(j\ k)|^2}{|Q'(j\ k)|^2} \quad \text{adjusted to minimum.}$$

It turns out that this operation does not alter the general form of the terms in the above matrix equation. Thus an iteration can be performed, and it is clear that if this iteration converges it must converge to the ideal least squares method approximation.

The general convergence properties of this method have not been investigated, but it is a practical experience that a good approximation may be obtained after, say, 3 iterations in many cases where the known function is related to a nuclear reactor transfer function.

The modifications introduced in the present version are as follows:

1) Instead of the weighting function $W(j\omega k) = 1/|Q'(j\omega k)|^2$ used in ref. 2 the user of this procedure may choose the alternative $1/|P'(j\omega k)|^2$. Assuming that the iterations converge, this leads to the fulfilment of the criterion

$$\sum_{k=1}^{m} |\mathcal{E}_k|^2 \quad \text{adjusted to minimum}$$

with $\mathcal{E}'k = \mathcal{E}k/G(j\omega k)$.

In other words, the least squares method is applied with respect to an error quantity which is very near to the relative error $\mathcal{E}k/F(j\omega k)$.

2) The elements of the matrix [A] are very often such that a direct calculation of the vector [X] leads to serios numerical deficiencies.

In order to avoid this, the single linear equations of the matrix equation are normalized by dividing with the square root of the sum of the squares of the left side coefficients.

3) As a criterion for stopping the iterations an error quantity is constructed after each iteration and compared with a quantity stated by the user. The former quantity is

$$\text{actual error} = \sqrt{(\sum (\frac{eak^2}{ak}) + epk^2)/m}$$

where eak/ak is the relative error in amplitude and epk the error in phase (in. radian measure) for the k th fit point.

It is noticed that this criterion does not influence the way the procedure works, except as to stop the calculations.

Use of the Algorithm:

The procedure call is COMPFIT (m, p, q, N, No, error, relative, drumomega, drumR, drumI, druma, drumb, drumc, drumA, singular);

The formal parameters have the following meaning:

integers:

m: number of fit points.
p: degree of numerator polynomial.
q: degree of denominator polynomial.
N: maximum number of iterations desired.
No: This formal parameter allows the user to break the series of iterations f.ex. for printing of preliminary results. If No is zero, which must always be the case the first time the procedure is called for solving a certain problem, the weighting function is equalled to unity at all fit points. If No is different from zero the procedure takes over from the drum store values of the weighting function thus assuming that these values have been constructed during a preceding procedure call and not later destroyed. It is noticed that the procedure, in counting the iterations performed, always assigns the number No+1 to the first iteration within a procedure call.